

# Initialization and Coordinate Optimization for Multi-way Matching

Da Tang<sup>1</sup> and Tony Jebara<sup>1,2</sup>

<sup>1</sup>Columbia University, New York, NY, USA

<sup>2</sup>Netflix Inc., Los Gatos, CA, USA

November 4, 2016

## Abstract

We consider the problem of consistently matching multiple sets of elements to each other, which is a common task in fields such as computer vision. To solve the underlying NP-hard objective, existing methods often relax or approximate it, but end up with unsatisfying empirical performances due to their inexact objectives. We propose a coordinate update algorithm that directly solves the exact objective. By using the pairwise alignment information to build an undirected graph and initializing the permutation matrices along the edges of its Maximum Spanning Tree, our algorithm successfully avoids bad local optima. Theoretically, with high probability our algorithm could guarantee to solve this problem optimally on data with reasonable noise. Empirically, our algorithm consistently and significantly outperforms existing methods on several benchmark tasks on real datasets.

## 1 INTRODUCTION

Given element sets  $X_1, \dots, X_n$  ( $n \geq 2$ ), the problem of finding consistent pairwise bijections between all pairs of them is known as multi-way matching. As a critical problem in computer science, it is widely used in many computer vision tasks, such as object recognition (Demirci et al., 2006; Yang et al., 2011), shape analysis (Petterson et al., 2009), and structure from motion (Dai et al., 2014; Simon et al., 2007; Roberts et al., 2011). It can also be applied to some other fields (e.g. multiple graph matching (Lacoste-Julien et al., 2006; Yan et al., 2013) and data source integration (Zhang et al., 2015)).

In most cases, people model the multi-way matching problem using a weighted multi-dimensional matching objective or its extension (e.g. the work (Pachauri et al., 2013; Yan et al., 2015; Zhou et al., 2015)). This objective is easy to solve when  $n = 2$ , where no consistency is required. However, as  $n \geq 3$ , it becomes extremely hard due to the combinatorial constraints for consistency. This objective is NP-hard to solve even when  $n = 3$  for the unweighted case (Ausiello et al., 2012; Kann, 1991). Hence, people designed many approximate methods, such as convex relaxation (Chen et al., 2014; Pachauri et al., 2013) and matrix decomposition (Yan et al., 2015; Zhou et al., 2015). These methods work well on datasets with little noise while they may not be stable on very noisy data since they are not solving the exact optimization objective due to their relaxation or approximation.

In this paper, we aim to find good algorithms to solve the exact objective for the weighted multi-dimensional matching problem. An intuitive method is to iteratively update the matching between the pair of sets  $(X_i, X_{i+1})$ 's for  $i = 1, \dots, n - 1$ , but that may produce significant errors once

---

E-mail: [datang@cs.columbia.edu](mailto:datang@cs.columbia.edu), [jebara@cs.columbia.edu](mailto:jebara@cs.columbia.edu)

we have one error pairwise matching (Le et al., 2007; Tsochantaridis et al., 2005; Volkovs and Zemel, 2012). Another idea is to just do coordinate updates on the objective since we could always solve weighted bipartite matching optimally, but that also depends significantly on initializations and may have very bad performances.

In this paper, we combine the above two ideas and design an effective method for the multi-way matching problem. We build an undirected graph with edge weights being all pairwise matching similarity values, and use its Maximum Spanning Tree (MST) to find a good order for computing  $n - 1$  pairwise matchings, in order to avoid bad local optima by using that matchings as the initialization for the coordinate updates. This seemingly simple idea has very desirable performance in practice and satisfactory theoretical guarantees. In real experiments, we could easily get good results on many well-known datasets. For instance, we could stably get **0%** error on the famous datasets **CMU House** and **CMU Hotel** for the task of stereo landmark alignments for  $m = 30$  points, which is not easy for existing algorithms. Theoretically, we not only guarantee that our algorithm solves the problem optimally when pairwise alignment methods work. We also guarantee optimality with high probability when a spanning tree on the noise parameter graph has small bottleneck weight (the largest weight in a spanning tree) and under some other mild assumptions.

## 2 CONSISTENT MATCHING FOR SETS OF ELEMENTS

We model the problem in the same way as Pachauri et al. (2013). Assume we have  $n$  element sets  $X_1, X_2, \dots, X_n$  where each  $X_i$  has  $m$  elements  $X_i = \{x_1^i, x_2^i, \dots, x_m^i\}$ . For any pair of element set  $(X_i, X_j)$ , we assume there is a bijection between their elements such that the element  $x_p^i$  is mapped with the element  $x_q^j$  if and only if these two elements are closely related to each other. For example,  $X_1, \dots, X_n$  could be  $n$  figures of the same object and each figure  $X_i$  has  $m$  pixels  $x_1^i, x_2^i, \dots, x_m^i$ . Since these figures capture the same object, the pixels of these figures may have some intercorrelations with each other and hence such kind of bijection between each pair of the figures exists.

Intuitively, this kind of bijections should be consistent with each other, that is, if any element  $x_p^i$  is mapped with some element  $x_q^j$  and the element  $x_q^j$  is mapped with some element  $x_r^k$ , then the element  $x_p^i$  should also be mapped with the element  $x_r^k$ . More specifically, given the element sets  $X_1, \dots, X_n$ , we are interested in finding consistent bijection  $\tau_{ij} : \{1, \dots, m\} \rightarrow \{1, \dots, m\}$  between each pair of element sets  $(X_i, X_j)$  such that:  $x_{\tau_{ij}(p)}^j$  is mapped with  $x_p^i$ ,  $\tau_{ii}$  is the identity transform,  $\tau_{ij} = \tau_{ji}^{-1}$  and  $\tau_{jk} \circ \tau_{ij} = \tau_{ik}$ , for any element sets  $X_i, X_j, X_k$  and any element  $x_p^i$ .

Achieving this goal is equivalent to reordering the elements in each element set  $X_i$  such that the elements after permutations will be corresponding to each other if they have the same element index. Mathematically, finding consistent bijection  $\tau_{ij}$  for each pair of element sets  $(X_i, X_j)$  is equivalent to finding a bijection  $\sigma_i : \{1, \dots, m\} \rightarrow \{1, \dots, m\}$  for each element set  $X_i$ , such that any element  $x_p^i$  is mapped with an element  $x_q^j$  if and only if  $\sigma_i(p) = \sigma_j(q)$ . We could easily see that these two kinds of mappings have the relations that  $\tau_{ij} = \sigma_j^{-1} \circ \sigma_i$  for any  $\tau_{ij}$ ,  $\sigma_i$  and  $\sigma_j$ .

In order to find this kind of mappings  $\sigma_i$ 's, Pachauri et al. (2013) proposed an objective function. Consider that we have a *similarity matrix*  $T_{ij} \in \mathbb{R}^{m \times m}$  for the each pair of sets  $(X_i, X_j)$ . The entry  $[T_{ij}]_{p,q}$  on the  $p^{th}$  row and the  $q^{th}$  column of  $T_{ij}$  represents the similarity level between the elements  $x_p^i$  and  $x_q^j$ . We require that, the closer these two elements are related to each other, the larger this entry is. To be symmetric, we also require  $T_{ij} = T_{ji}^T$  for any pair of  $(T_{ij}, T_{ji})$ . Moreover, we also constrain each  $[T_{ij}]_{p,q}$  to be in the range  $[0, 1]$  such that we are treating all pairs of elements fairly. Ideally, the elements  $x_p^i$  and the elements  $x_q^j$  are perfectly related to each other if and only if  $[T_{ij}]_{p,q} = 1$ , while they are totally not related to each other if  $[T_{ij}]_{p,q} = 0$ . Pachauri et al. (2013)

aimed to find mappings  $\sigma_1, \dots, \sigma_n$  by solving the following optimization problem:

$$\max_{\sigma_1, \dots, \sigma_n} \sum_{i=1}^n \sum_{j=1}^n \langle P(\sigma_j^{-1} \circ \sigma_i), T_{ij} \rangle \quad (1)$$

Where  $P(\sigma) \in \mathbb{R}^{m \times m}$  is the permutation matrix such that

$$[P]_{p,q} = \begin{cases} 1 & \text{if } \sigma(p) = q \\ 0 & \text{otherwise.} \end{cases}$$

Notice that all permutation matrices are orthogonal matrices and  $P(\sigma_j^{-1} \circ \sigma_i) = P(\sigma_i)^{-1} P(\sigma_j)$  for any mappings  $\sigma_i$  and  $\sigma_j$ . Denote the set of all  $m \times m$  permutation matrices as  $\mathcal{P}_m$ , we could also write the objective function in Equation (1) as

$$\max_{A_1, \dots, A_n \in \mathcal{P}_m} \sum_{i=1}^n \sum_{j=1}^n \text{tr}(A_i T_{ij} A_j^\top) \quad (2)$$

since  $A_i^\top = A_i^{-1}$ , where  $A_i = P(\sigma_i)$  is a permutation matrix for the element set  $X_i$ . Note that the solution for this optimization problem is not unique (in fact, it has at least  $m!$  different tuples of solutions) since  $(A_1, \dots, A_n) = (P\hat{A}_1, \dots, P\hat{A}_n)$  is an optimal solution if  $(A_1, \dots, A_n) = (\hat{A}_1, \dots, \hat{A}_n)$  is, for any permutation matrix  $P \in \mathcal{P}_m$ .

A naive but intuitive method for solving this problem is by recovering  $A_1, \dots, A_n$  from the equations  $P_{ij} = A_i^\top A_j$ , where  $P_{ij} = \underset{P \in \mathcal{P}_m}{\text{argmax}} \text{tr}(P^\top T_{ij})$ , for all  $i, j \in \{1, \dots, n\}$ . We call this method as the *Pairwise Alignment* method. But this method do not always work since these matrices  $P_{ij}$ 's may not be consistent with each other (i.e. don't always satisfy  $P_{ij}P_{jk} = P_{ik}$ ) and we may not have a solution for  $(A_1, \dots, A_n)$ . In the next section, we will discuss a useful algorithm for solving this optimization problem.

### 3 COORDINATE OPTIMIZATION WITH SMART INITIALIZATION

The optimization problem in Equation (1) is basically a maximum weighted  $n$ -dimensional matching problem. We know that, even if we only have  $n = 3$  and require the similarity matrices  $T_{ij}$ 's to have values of only 0 or 1 (i.e. the unweighted 3-dimensional matching problem), it is still NP-hard and could not be approximated in a ratio of  $\frac{3}{2}$  in polynomial time (Ausiello et al., 2012; Kann, 1991). Therefore, we do not aim at finding solutions to this optimization problem for all possible values of  $T_{ij}$ 's. To solve this problem approximately, Pachauri et al. (2013) proposed an eigenvalue decomposition-based method by first relaxing the combinatorial optimization objective into a continuous one and then do the rounding using Kuhn-Munkres algorithm (Kuhn, 2010). However, Pachauri et al. (2013) could only guarantee a good performance of their algorithm when every similarity matrix  $T_{ij}$  is very close to the ground truth permutation matrices  $P(\tilde{\sigma}_j^{-1} \circ \tilde{\sigma}_i)$  (denote  $\tilde{\sigma}_1, \dots, \tilde{\sigma}_n$  to be the ground truth mappings we want to find). Unfortunately, it is not always in this case in practice. In this section, we will discuss a new method of solving this problem via coordinate ascent.

### 3.1 Coordinate Ascent for Permutations

Consider the objective function in Equation (2), for each permutation matrix  $A_i$ , since  $\text{tr}(A_i T_{ii} A_i^\top) = \text{tr}(A_i^\top A_i T_{ii}) = \text{tr}(T_{ii})$  is a constant, and  $\text{tr}(A_i T_{ij} A_j^\top) = \text{tr}(A_j T_{ij}^\top A_i^\top) = \text{tr}(A_j T_{ji} A_i^\top)$  for any permutation matrix  $A_j$ , we know that, if we fix all of the other permutation matrices  $A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_n$ , then the maximization problem becomes

$$\begin{aligned} & \underset{A_i \in \mathcal{P}_m}{\text{argmax}} \sum_{i=1}^n \sum_{j=1}^n \text{tr}(A_i T_{ij} A_j^\top) \\ &= \underset{A_i \in \mathcal{P}_m}{\text{argmax}} \text{tr}(A_i^\top \sum_{1 \leq j \leq n, i \neq j} A_j T_{ij}). \end{aligned} \quad (3)$$

The optimization in Equation (3) could be solved in polynomial time (for example, through Kuhn–Munkres algorithm (Kuhn, 2010) in time  $O(m^3)$ ). Hence, a naive coordinate algorithm is very easy to be derived: We initialize the permutation matrices  $A_1, \dots, A_n$  (either randomly or deterministically). During each iteration, we randomly pick  $i \in \{1, \dots, n\}$  and update  $A_i$  according to Equation (3), until convergence. Unfortunately, a bad initialization for this algorithm may lead to an unsatisfying local optimum (we will show details in Section 4.2). But better performances may be obtained if we could use the pairwise alignment information to construct a good initialization, which will be discussed in the next section.

### 3.2 Adding MST-based initializations

According to the discussion above, obtaining a good initialization for this coordinate update process in Equation (1) is very important. Intuitively, consider a single term  $\text{tr}(A_i T_{ij} A_j^\top)$  in our objective function in Equation (2), since we want to maximize that objective, we probably think that the values for  $A_i$  and  $A_j$  are relatively “convincing” if the term  $\text{tr}(A_i T_{ij} A_j^\top)$  is very large. Define

$$f(T_{ij}) := \max_{P \in \mathcal{P}_m} \text{tr}(P^\top T_{ij}) = \max_{A_i, A_j \in \mathcal{P}_m} \text{tr}(A_i T_{ij} A_j^\top) \quad (4)$$

for each  $T_{ij}$ . We tend to believe that, if we have an initialization  $A_i = \hat{A}_i$  and  $A_j = \hat{A}_j$  for some permutation matrices  $\hat{A}_i, \hat{A}_j \in \mathcal{P}_m$  and  $\text{tr}(\hat{A}_i T_{ij} \hat{A}_j^\top)$  close or equal to  $f(T_{ij})$ , then this initialization on these two matrices are convincing.

With the above idea, we come up with an initialization method that will first initialize the matrices  $A_i, A_j$  if the corresponding  $f(T_{ij})$  is large. To achieve this goal, we could build an undirected graph  $G = (V, E)$ , where each element set  $X_i$  corresponds to one vertex  $v_i \in V$  and each pair of element sets  $(X_i, X_j)$  ( $i \neq j$ ) has an edge  $(v_i, v_j) \in E$  with weight  $f(T_{ij}) = f(T_{ji})$ . We first find a Maximum Spanning Tree  $T = (V, E')$  of the Graph  $G$ . Then, we initialize the matrices  $A_1, \dots, A_n$  along the edges in  $E'$  as follows: Initially we have  $n$  sets  $S_1, \dots, S_n$  of vertices, each contains one vertex in  $V$ . Then for every edge in each edge  $(v_i, v_j) \in E'$ , we try to “combine” them together and use the similarity matrix  $T_{ij}$  to optimize the permutation matrices correspond to the vertices in the sets contain  $v_i$  and  $v_j$ . Details are shown in Algorithm Box 1.

As in this algorithm, the first 8 lines uses the Maximum Spanning Tree to initialize the permutation matrices  $A_1, \dots, A_n$  in a smart way. Intuitively, if we want to use the method of iteratively “combining” the vertices in  $V$  to do the initialization as we do in Algorithm 1, we want each edge  $(v_i, v_j)$  that is used should have the corresponding  $f(T_{ij})$  value relatively large. Hence, using the edges in a Maximum Spanning Tree of  $G$  is a good choice. And the last 4 lines in this algorithm is just a regular coordinate update process. We hope the smart initialization will make a good start for the coordinate updates and hence we could get a good solution for the permutation matrices  $(A_1, \dots, A_n)$ .

---

**Algorithm 1** MST-based coordinate updates for Multi-way matching

---

**Input:**

The similarity matrices  $T_{ij}$ 's ( $i, j \in \{1, \dots, n\}$ ).

**Output:**

The permutation matrices  $A_i$ 's ( $i \in \{1, \dots, n\}$ ).

- 1: Construct the Graph  $G = (V, E)$  as in the Section 3.2, compute a Maximum Spanning Tree  $T = (V, E')$  of  $G$ ;
  - 2: Initialize  $S_i \leftarrow \{v_i\}$  for each  $v_i \in V$ ; For each  $A_i$ , initialize it to be any permutation matrix in  $\mathcal{P}_m$ ;
  - 3: **for** each edge  $(v_i, v_j) \in E'$  **do**
  - 4:   Compute  $\hat{P} = \underset{P}{\operatorname{argmax}} \operatorname{tr}(P^\top A_i T_{ij} A_j^\top)$ ;
  - 5:   Update  $A_{j'} \leftarrow \hat{P} A_{j'}$  for each  $v_{j'} \in S_j$ ;
  - 6:   Let  $S' = S_i \cup S_j$ ;
  - 7:   Update  $S_k \leftarrow S'$  for each  $v_k \in S'$ ;
  - 8: **end for**
  - 9: **while** Not converged **do**
  - 10:   Randomly pick  $i \in \{1, \dots, n\}$ ;
  - 11:   Update  $A_i$  according to Equation (3);
  - 12: **end while**
- 

### 3.3 Analysis of the Coordinate Update

#### 3.3.1 Analysis under the case without noise

Then let us analyze how does Algorithm 1 behave. First, we consider a simple case where we could guarantee to use the Pairwise Alignment method to find consistent permutation matrix  $P_{ij} = f(T_{ij})$  for each pair of element sets  $(X_i, X_j)$ , i.e.  $P_{ij}P_{jk} = P_{ik}$  for all  $i, j, k \in \{1, \dots, n\}$  (here the word "guarantee" means that the maximum value of  $\operatorname{tr}(P^\top T_{ij})$  should be achieved for only one permutation matrix  $P$  and hence  $f(T_{ij})$  is well defined, for each similarity matrix  $T_{ij}$ ). Then definitely we could recover the optimal  $(A_1, \dots, A_n)$  from those  $P_{ij}$  matrices by setting  $A_i = P_{1i}$  for each  $A_i$  since each single term  $\operatorname{tr}(A_i T_{ij} A_j^\top) = \operatorname{tr}(A_j^\top A_i T_{ij}) = \operatorname{tr}(P_{ij}^\top T_{ij})$  in the objective function in the Equation (2) is maximized.

Then, what's our Algorithm 1's behavior under this constraint? Could we guarantee that our algorithm can also recover all of the matrices  $A_i$ 's optimally? The answer is YES. We have the following theorem.

**Theorem 1.** *If we could guarantee to recover consistent permutation matrix  $P_{ij} = f(T_{ij})$  for each pair of element sets  $(X_i, X_j)$  using the Pairwise Alignment method, then we could also guarantee that Algorithm 1 can solve the optimization problem in Equation 2 optimally.*

The proof is in Section A of the supplementary material. From Theorem 1, we know that Algorithm 1 is at least as good as the Pairwise Alignment method on the ability of optimally solving the consistent matching problem. Moreover, the optimal case in Theorem 1 contain all cases that Pachauri et al. (2013) claimed that they could solve optimally, and we could guarantee to solve optimally on much more cases than their method.

### 3.3.2 Analysis under the case with noise

We are more interested in the case where the matrices  $T_{ij}$ 's are not perfect and have some noise. If we denote the optimal solution for the optimization problem in Equation (2) as  $(A_1, \dots, A_n) = (\hat{A}_1, \dots, \hat{A}_n)$ , then ideally the best data we could have for each  $T_{ij}$  should be  $T_{ij} = \hat{T}_{ij} := \hat{A}_i^\top \hat{A}_j$ . If we are in this case where  $T_{ij} = \hat{T}_{ij}$  for each  $i, j \in \{1, \dots, n\}$ , then obviously from Theorem 1 we know that Algorithm 1 solves this optimization problem optimally.

Then, we want to know what if the similarity matrices have some noise and  $T_{ij}$  is not perfectly  $\hat{T}_{ij}$  for some (or all) of the  $T_{ij}$ 's? We could assume that each  $T_{ij}$  to be a random matrix near  $\hat{T}_{ij}$  and try to analyze Algorithm 1's performance. Here we only need to model the matrices  $T_{ij}$  where  $i \neq j$  since we will never use the matrices  $T_{ii}$ 's in Algorithm 1. Since usually we want each  $T_{ij}$  to only have entries in the range  $[0, 1]$ , we could model its entries as follows:

$$[T_{ij}]_{p,q} = \begin{cases} 1 - Z_{ijpq}^2 & \text{if } i < j \text{ and } [\hat{T}_{ij}]_{p,q} = 1 \\ Z_{ijpq}^2 & \text{if } i < j \text{ and } [\hat{T}_{ij}]_{p,q} = 0 \\ [T_{ji}]_{q,p} & \text{if } i > j. \end{cases} \quad (5)$$

Where  $Z_{ijpq} \sim \mathcal{N}(0, \eta_{ij})$ 's are independent Gaussian random variables for any  $1 \leq i < j \leq n$  and  $p, q \in \{1, \dots, m\}$ . Here we assume that different  $T_{ij}$  matrices may have different variance parameters  $\eta_{ij}$ 's since we may have different noisy levels for different pairs of element sets. Also, we require  $\eta_{ij} \leq O(1)$  for each  $\eta_{ij}$  since we want the similarity matrices to only have entries in  $[0, 1]$ . Notice that we will still have  $T_{ij} = T_{ji}^\top$  for all  $i \neq j$  under the model in Equation (5). We have the following theorem:

**Theorem 2.** *If the following conditions hold, then for sufficiently large  $n$  and  $m$ , with probability  $1 - o(1)$  Algorithm 1 could guarantee to find an optimal solution for the optimization problem in Equation (2):*

- $n \geq 20 \ln m$ , and there exists  $\gamma > 0$  such that  $n \leq m^\gamma$ .
- The bottleneck weight of the Minimum Bottleneck Spanning Tree (Polzin and Daneshmand, 2003) of  $G'$  is at most  $\frac{1}{4(3+\gamma)\ln m + 4}$ . Here  $G' = (V', E'')$  is an undirected weighted graph, where there is a vertex  $v'_i \in V'$  for each element set  $X_i$  and there is an edge  $(v'_i, v'_j) \in E''$  with weight  $\eta_{ij}$ .
- $\max_{1 \leq i < j \leq n} \eta_{ij} \leq \frac{1}{3}$ .

The proof is in Section B of the supplementary material. Here the Minimum Bottleneck Spanning Tree of a graph  $G'$  means a spanning tree of  $G'$  with its largest edge weight being minimized. It seems that our algorithm could work very well as  $n$  and  $m$  are both very large and there exists a spanning tree of Graph  $G'$  with all edge weights no more than  $O(\frac{1}{\log m})$ . In the proof for this theorem we will show that, we could use the Pairwise Alignment method to solve the optimization problem optimally with high probability if all edges of  $G'$  have weights no more than  $O(\frac{1}{\log m})$ , which is the same asymptotically with our bottleneck weight bound but it is for all edges in  $E''$ . So our algorithm could accept much more kinds of data and get an optimal solution with high probability.

### 3.4 Practical Improvements

In Section 3.2 and 3.3, we introduced our algorithm and discussed its theoretical guarantees. However, to potentially make Algorithm 1 behave better in practice, we could make some improvements on it so that hopefully we could get better empirical performances.

### 3.4.1 Use a good MST edge order

In Algorithm 1, we do initializations by handling the edges of the Maximum Spanning Tree  $T$ . It is reasonable that running the algorithm in a good order for the edges may be beneficial. In this section, we propose two kinds of order that have been found working well in practice, the Prim's order and the Kruskal's order.

As seen in line 5 of Algorithm 1, we will need to update  $|S_j|$  different permutation matrices in one step. Though we have proved that this algorithm is working well on many cases, we still believe that it may be better if we could be more "cautious", that is, to update less number of permutation matrices at each iteration. Therefore, we could use Prim's algorithm (Ahuja, 1988) to compute the Maximum Spanning Tree and handle the edges in the order that we get them during the process of Prim's algorithm. Since in this case there is only one vertex in the set  $|S_j|$  each time, we only need to update one permutation matrix at each iteration. We call this order the *Prim's order*.

Meanwhile, the edge weights are also very important for initialization. As discussed before in Section 3.3.1, we believe more in those edges  $(v_i, v_j)$ 's whose weights  $f(T_{ij})$ 's are larger. Hence, we could handle those edges that we trust more first. To achieve that goal, we could just simply handle all of the edges in the descending order of their weights. This is exactly the same with the edge order that we get in a run of Kruskal's algorithm (Ahuja, 1988). And hence we call it the *Kruskal's order*.

### 3.4.2 Combine initializations with coordinate optimizations

In Algorithm 1, we have a coordinate update process after initialization. However, we might have some probability going to some bad points during the initialization as well. Hence, it is worthy adding a coordinate update process right after each iteration of initialization that may potentially fix some errors the algorithm probably make during that iteration. Since at each iteration we treating with the vertices in the set  $S'$  (line 6 of Algorithm 1), we could just do a coordinate update on those corresponding permutation matrices  $A_k$ 's where  $v_k \in S'$  as:

$$A_k = \underset{P}{\operatorname{argmax}} \operatorname{tr}(P^\top \sum_{v_{k'} \in S', k' \neq k} A_{k'} T_{kk'}). \quad (6)$$

By adding this kind of intermediate update steps, we no longer need to have the final coordinate update process since the additional coordinate updates after the last iteration of initialization have taken that role.

### 3.4.3 The overall algorithm

By adding the heuristics mentioned above, our algorithm is updated and details are as follows:

We hope this algorithm could work better in real experiments. Using similar ideas as the proof for Theorem 1, it is easy to show that the Algorithm 2 is still at least as good as the Pairwise Alignment method on solving the optimization problem optimally:

**Theorem 3.** *If we could guarantee to recover consistent permutation matrix  $P_{ij} = f(T_{ij})$  for each pair of element sets  $(X_i, X_j)$  using the Pairwise Alignment method, then we could also guarantee that Algorithm 2 can solve the optimization problem in Equation 2 optimally.*

## 4 EXPERIMENTS

In this section, we will show how our Algorithm 2 behaves in practice. We try it on various datasets in computer vision, the major application field for our algorithm. For each dataset, we compare our

---

**Algorithm 2** Improved MST-based coordinate updates for Multi-way matching

---

**Input:**

The similarity matrices  $T_{ij}$ 's ( $i, j \in \{1, \dots, n\}$ ).

**Output:**

The permutation matrices  $A_i$ 's ( $i \in \{1, \dots, n\}$ ).

- 1: Construct the Graph  $G = (V, E)$  as in the Section 3.2, Compute a Maximum Spanning Tree  $T = (V, E')$  of  $G$ ;
  - 2: Sort the edges in  $E'$  in the Prim's order or the Kruskal's order as discussed in Section 3.4;
  - 3: Initialize  $S_i \leftarrow \{v_i\}$  for each  $v_i \in V$ ; For each  $A_i$ , initialize it to be any permutation matrix in  $\mathcal{P}_m$ ;
  - 4: **for** each edge  $(v_i, v_j) \in E'$  **do**
  - 5:   Compute  $\hat{P} = \underset{P}{\operatorname{argmax}} \operatorname{tr}(P^\top A_i T_{ij} A_j^\top)$ ;
  - 6:   Update  $A_{j'} \leftarrow \hat{P} A_{j'}$  for each  $v_{j'} \in S_j$ ;
  - 7:   Let  $S' = S_i \cup S_j$ ;
  - 8:   Update  $S_k \leftarrow S'$  for each  $v_k \in S'$ ;
  - 9:   **while** Not converged **do**
  - 10:     Randomly pick  $v_k \in S'$ ;
  - 11:     Update  $A_k$  according to Equation (6);
  - 12:   **end while**
  - 13: **end for**
- 

algorithm with the Permutation Synchronization algorithm that Pachauri et al. (2013) proposed, which is the state-of-the-art method that has the same objective with us.

## 4.1 PCA Reconstruction for Digits

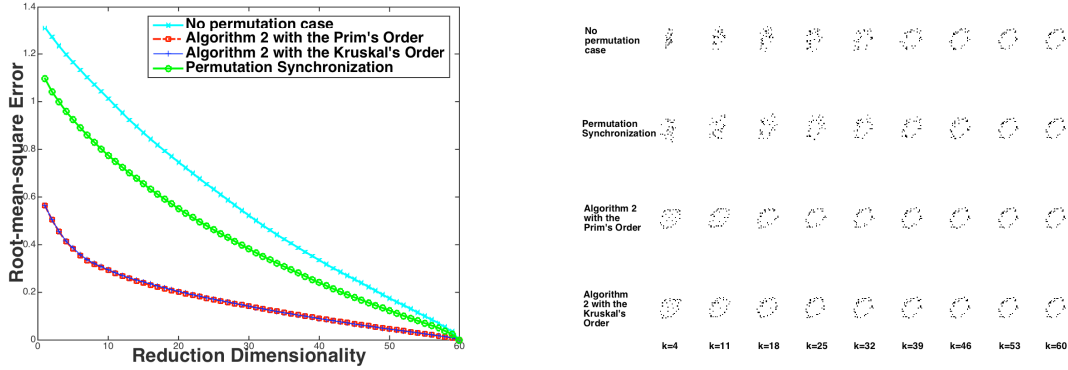
Our first experiment is on figure compression and recovery. We use the **MNIST** dataset (LeCun et al., 1998), which contains 70000 figures of handwriting digits. Each figure is a handwriting version of one digit in  $\{0, \dots, 9\}$  and each digit has roughly the same number of figures in this dataset.

In our experiments, we randomly select  $n = 100$  figures  $F_1, \dots, F_n$  from the dataset, where each digit has roughly  $\frac{n}{10}$  figures. We want to see if our algorithm could make the compression storage of these figures better. To make the dataset fit our problem's input form, we randomly select  $m = 30$  white pixels (the digits are white and the background color is black in the dataset) for each figure  $F_i$  and build an element set  $X_i = \{x_j^i = (a_j^i, b_j^i) : j \in \{1, \dots, m\}\}$ , where  $(a_j^i, b_j^i)$  is the coordinate of the  $j^{th}$  selected pixel of figure  $F_i$ .

We want to see if we could improve the compression of these element sets  $X_1, \dots, X_n$  by using our algorithm. An easy and reasonable way to do the compression is through Principle Component Analysis (PCA). We can view each  $X_i$  as a matrix  $Y_i \in \mathbb{R}^{m \times 2}$  where each the  $j^{th}$  row of  $Y_i$  is just  $(a_j^i, b_j^i)$  and we could try to use PCA to compressively store the vectors  $\operatorname{vec}(Y_1), \dots, \operatorname{vec}(Y_n) \in \mathbb{R}^{2m}$ . We could choose any number  $k \in \{1, \dots, \min(n, 2m)\}$  and use PCA to reduce the dimensionality of these vectors to  $k$  in the compression process, getting a  $k$ -dimensional orthogonal basis and a mean vector in  $\mathbb{R}^{2m}$ . In the recovery process, we could use them to reconstruct the original data. Since each of these figures is a digit, we believe that, for many pairs of the figures, there should be a bijection correspondence relationship between their selected pixels due to their similar geometry structures. Our goal is to reduce the reconstruction error by using our algorithm to reorder the elements in each element set  $X_i$  before compression.

The experiment results are shown in Figure 1. Here we use the Radial basis function kernel





(a) The PCA reconstruction error of two versions of our Algorithm 2 compared to two baseline methods. The horizontal axis represents the reduction dimensionality  $k$ . The performances of our methods are very close to each other, which are much better than the baseline ones.

(b) The reconstructed results of a figure of digit 0 by all methods. For each figure, we show the reconstructed figures with  $k = 4, 11, 18, \dots, 60$ , respectively. We can see that our methods recover the figure very close to a digit 0 even at  $k = 4$ , but the other two methods could not.

Figure 1: Results for the PCA reconstruction experiment.

(RBF kernel) to compute the similarity matrices  $T_{ij}$ 's. More specifically,  $[T_{ij}]_{p,q} = \exp(-\frac{\|x_p^i - x_q^j\|^2}{2\sigma^2})$  where  $\sigma$  is a parameter. This function is suitable for our settings, since we expect  $[T_{ij}]_{p,q}$  to be close to 1 when  $x_p^i$  and  $x_q^j$  are close to each other, and to be close to 0 otherwise. We compare the two versions of our algorithm (the Prim's Order and the Kruskal's Order) with two baseline cases: 1. We don't reorder the pixels at all (the no permutation case); 2. We use the Permutation Synchronization algorithm to reorder the pixels before compression. For each method, we select a best  $\sigma$  value for it, and consider its performance under the selected parameter. Figure 1 shows the reconstruction errors and the recovered digits we get from these four methods and we could see that both versions of our algorithm outperform the two baseline methods.

## 4.2 Stereo Landmark Alignments

Table 1: Average Error Rates of alignments for the datasets House, Hotel and Building

TASK	PERM-SYNC	PRIM'S ORDER	KRUSKAL'S ORDER
CMU House RBF	$(22.61 \pm 7.49)\%$	<b><math>(0.00 \pm 0.00)\%</math></b>	<b><math>(0.00 \pm 0.00)\%</math></b>
CMU House Alignment	$(4.71 \pm 1.98)\%$	<b><math>(0.81 \pm 0.96)\%</math></b>	$(1.89 \pm 2.21)\%$
CMU Hotel RBF	$(18.63 \pm 2.90)\%$	<b><math>(0.00 \pm 0.00)\%</math></b>	<b><math>(0.00 \pm 0.00)\%</math></b>
CMU Hotel Alignment	$(5.22 \pm 2.55)\%$	<b><math>(3.59 \pm 0.75)\%</math></b>	$(4.20 \pm 0.71)\%$
Building RBF	$(86.71 \pm 3.36)\%$	<b><math>(49.87 \pm 0.24)\%</math></b>	$(50.39 \pm 0.25)\%$
Building Alignment	$(50.49 \pm 1.09)\%$	<b><math>(48.52 \pm 0.50)\%</math></b>	$(48.61 \pm 0.52)\%$

The second question we focus on is stereo matching, which is to align pixels of multiple figures of

a single object. For this experiment, we have 2 datasets, the **CMU House** dataset and the **CMU Hotel** dataset. Each of them contains  $n$  figures of a same toy house ( $n = 111$  for the CMU House dataset and  $n = 101$  for the CMU Hotel dataset). For each of the two toy houses, we have  $m = 30$  landmark points selected, and each of the  $n$  figures contains a different view of these  $m$  landmark points (i.e. each figure has  $m$  points represent them). Our goal is to find a consistent mapping between all of these points that maps points correspond to a same landmark together.

The element sets are constructed by extracting visual features. Same as the first experiment, we use the RBF kernel to compute the similarity matrices. We call experiment tasks under this setting as *CMU House RBF* and *CMU Hotel RBF*, respectively. In addition to this setting, since we have ground truth alignments for these two datasets, it is reasonable to use some local alignments between pairs of figures to construct the similarity matrices. Hence, we also use the outputs of the Pairwise Alignment method to be the similarity matrices, and call the corresponding tasks as *CMU House Alignment* and *CMU Hotel Alignment*. We compute the average error rates of all pairs of element sets as the criteria for performance.

The experiment results are shown in the first four lines of Table 1. We test the two versions (the Prim’s Order and the Kruskal’s Order) of Algorithm 2 and the baseline algorithm Permutation Synchronization. Here we run 10 times for each task and we first randomly order the figures and the pixels before running the algorithms each time, and hence the performances may be stochastic. We can see that both of our methods could stably solve this problem with **100%** accuracy on the RBF tasks, while the baseline method behaves much worse. For the Alignment tasks, the baseline method’s behaviors get better since it tend to excel in that kinds of inputs. But our methods could still outperform it, especially the Prim’s Order version.

Notice that the initialization part of our algorithms are very important. Without the smart initialization technique, the pure coordinate updates could behave much more randomly. For instance, in the CMU House RBF task, we will get an average error rate of  $(3.90 \pm 2.63)\%$  if we just randomly initialize all permutation matrices, while we could always get 0% error rates by using our MST-based initialization technique.

### 4.3 Repetitive Structures of Key Points

Instead of stereo matching, we are also interested in matching figures with complicated geometry ambiguities. For instance, figures with frequent repetitive structures are extremely hard to be handled even if we use some sophisticated features such as SIFT (Pachauri et al., 2013). In this experiment, we use the **Building** dataset (Roberts et al., 2011) which has such kind of structures. Pachauri et al. (2013) used this dataset and hand annotated 25 “similar looking” landmark points in the scene across the dataset. We use their hand annotated version and select  $n = 14$  figures from it for evaluation. However, in each figure, we have  $m = 28$  key points, and we do not always have all of the 25 landmark points in each scene. Hence, there are many useless key points in each figure, which makes the task of finding true consistent alignment extremely hard.

Same as the Stereo Landmark Alignments test in Section 4.2, we still have 2 settings, the RBF setting and the Alignment setting, for this experiment. The performances of our Algorithm 2 compared to the baseline algorithm Permutation Synchronization is shown on the last two rows of Table 1. This task is much more difficult than stereo matching and we don’t expect a very high accuracy. As can be seen there, both of our methods could outperform the baseline algorithm in both settings, especially in the RBF one. Furthermore, the alignments we get for each setting and each algorithm are shown in Figure 2. All of those 6 groups of alignments are between a same pair of

---

Datasets URL: <http://vasc.ri.cmu.edu/idb/html/motion/>.



(a) The Permutation Synchronization method under the RBF setting



(b) Algorithm 2 with the Prim's Order under the RBF setting



(c) Algorithm 2 with the Kruskal's Order under the RBF setting



(d) The Permutation Synchronization method under the Alignment setting



(e) Algorithm 2 with the Prim's Order under the Alignment setting



(f) Algorithm 2 with the Kruskal's Order under the Alignment setting

Figure 2: Key Points Alignment for the Building dataset. The green lines are the ground truth alignments while the red lines are the computed alignments. Green lines represent the ground truth matchings and the red lines represent the matchings that the algorithms obtained. Less green lines being exposed means better performance.

figures. It can be seen that, even we have strong repetitive structure and large noise in the datasets, our Algorithm 2 could still be stable and get satisfactory consistent matchings.

## 5 CONCLUSION

Straightforward coordinate update for the multi-way matching objective is not stable and may behave very poorly. However, if we could have a good initialization for that update, it will not get stuck at bad local optima. By combining the traditional coordinate ascent algorithm with iterative initializations along the edges of the Maximum Spanning Tree of the graph with edge weights being the pairwise matching similarity values, we have much more accurate and stable empirical results on various different computer vision tasks (figure reconstruction, stereo landmark alignments as well as matching key points with repetitive structures) compared with the baseline methods. In addition to that, from the theoretical analysis we know that we do not need all of the noise parameters to be small to get a perfect optimization result with high probability, but only require a spanning tree on the noise parameter graph to have a small bottleneck edge weight as long as some basic constraints are satisfied.

For future work, we aim to improve our consistent matching algorithm as a great feature learning tool. We expect our method could be used for preprocessing datasets so that they can be better fit for common machine learning tasks, such as classification, regression and clustering.

## Acknowledgements

Work supported in part by DARPA N66001-15-2-4026, N66001-15-C-4032 and NSF III-1526914, IIS-1451500, CCF-1302269.

## References

- Ravindra K Ahuja. *Network flows*. PhD thesis, TECHNISCHE HOCHSCHULE DARMSTADT, 1988.
- Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media, 2012.
- Yuxin Chen, Leonidas J Guibas, and Qi-Xing Huang. Near-optimal joint object matching via convex relaxation. *arXiv preprint arXiv:1402.1473*, 2014.
- Yuchao Dai, Hongdong Li, and Mingyi He. A simple prior-free method for non-rigid structure-from-motion factorization. *International Journal of Computer Vision*, 107(2):101–122, 2014.
- M Fatih Demirci, Ali Shokoufandeh, Yakov Keselman, Lars Bretzner, and Sven Dickinson. Object recognition as many-to-many feature matching. *International Journal of Computer Vision*, 69(2): 203–222, 2006.
- Heinrich Dörrie. *100 great problems of elementary mathematics*. Courier Corporation, 2013.
- Viggo Kann. Maximum bounded 3-dimensional matching is max snp-complete. *Information Processing Letters*, 37(1):27–35, 1991.
- Harold W Kuhn. The hungarian method for the assignment problem. In *50 Years of Integer Programming 1958-2008*, pages 29–47. Springer, 2010.
- Simon Lacoste-Julien, Ben Taskar, Dan Klein, and Michael I Jordan. Word alignment via quadratic assignment. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 112–119. Association for Computational Linguistics, 2006.
- Beatrice Laurent and Pascal Massart. Adaptive estimation of a quadratic functional by model selection. *Annals of Statistics*, pages 1302–1338, 2000.
- Quoc V Le, Alex J Smola, and Svn Vishwanathan. Bundle methods for machine learning. In *Advances in neural information processing systems*, pages 1377–1384, 2007.
- Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- Deepti Pachauri, Risi Kondor, and Vikas Singh. Solving the multi-way matching problem by permutation synchronization. In *Advances in neural information processing systems*, pages 1860–1868, 2013.
- James Petterson, Jin Yu, Julian J McAuley, and Tibério S Caetano. Exponential family graph matching and ranking. In *Advances in Neural Information Processing Systems*, pages 1455–1463, 2009.

- Tobias Polzin and Siavash Vahdati Daneshmand. On steiner trees and minimum spanning trees in hypergraphs. *Operations Research Letters*, 31(1):12–20, 2003.
- Richard Roberts, Sudipta N Sinha, Richard Szeliski, and Drew Steedly. Structure from motion for scenes with large duplicate structures. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3137–3144. IEEE, 2011.
- Ian Simon, Noah Snavely, and Steven M Seitz. Scene summarization for online image collections. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.
- Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(Sep):1453–1484, 2005.
- Maksims Volkovs and Richard S Zemel. Efficient sampling for bipartite matching problems. In *Advances in Neural Information Processing Systems*, pages 1313–1321, 2012.
- Junchi Yan, Yu Tian, Hongyuan Zha, Xiaokang Yang, Ya Zhang, and Stephen M Chu. Joint optimization for consistent multiple graph matching. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1649–1656, 2013.
- Junchi Yan, Hongteng Xu, Hongyuan Zha, Xiaokang Yang, Huanxi Liu, and Stephen Chu. A matrix decomposition perspective to multiple graph matching. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 199–207, 2015.
- Allen Y Yang, Subhransu Maji, C Mario Christoudias, Trevor Darrell, Jitendra Malik, and S Shankar Sastry. Multiple-view object recognition in smart camera networks. In *Distributed Video Sensor Networks*, pages 55–68. Springer, 2011.
- Duo Zhang, Benjamin IP Rubinstein, and Jim Gemmell. Principled graph matching algorithms for integrating multiple data sources. *IEEE Transactions on Knowledge and Data Engineering*, 27(10):2784–2796, 2015.
- Xiaowei Zhou, Menglong Zhu, and Kostas Daniilidis. Multi-image matching via fast alternating minimization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4032–4040, 2015.

## SUPPLEMENTARY MATERIAL

In this supplementary material, we will prove the Theorem 1 and 2 in the main paper. Theorem 3 could be proved using almost the same idea with our proof for Theorem 1 and hence its proof is omitted.

### A. Proof for Theorem 1

*Proof.* Since we have mentioned that, under the case of this theorem the matrices  $P_{ij}$  satisfy the sum  $\sum_{i=1}^n \sum_{j=1}^n \text{tr}(P_{ij}^\top T_{ij})$  reaches the optimal value for the objective in Equation (2) in the main paper, it is sufficient to show that, the matrices  $A_1, \dots, A_n$  returned by Algorithm 1 should satisfy  $P_{ij} = A_i^\top A_j$  for each  $P_{ij}$ . We first show that, before the coordinate update part (line 9 to 12) of the Algorithm 1, we have already make the matrices  $A_1, \dots, A_n$  having the property of  $P_{ij} = A_i^\top A_j$  for each  $P_{ij}$ . We will use induction to prove that, after each iteration during the initialization part of the Algorithm 1, for any set  $S_k$  and any  $v_i, v_j \in S_k$ , we have  $P_{ij} = A_i^\top A_j$ .

1. Initially (after the  $0^{th}$  iteration), each set  $S_k$  only contain one vertex  $v_k$ . Since  $P_{ii} = I = A_k^\top A_k$ , the induction assumption is correct.
2. Assume the induction assumption is correct after the  $t^{th}$  iteration ( $t \geq 0$ ). For the  $(t+1)^{th}$  iteration, denote the edge we use in this iteration as  $(v_i, v_j)$ , then from the algorithm we know that the matrix  $\hat{P} = \arg\max_P \text{tr}(P^\top A_i T_{ij} A_j^\top) = A_i P_{ij} A_j^\top$  for the old values of  $A_i$  and  $A_j$ .

Therefore, after the update on line 5, we will get  $P_{ij} = A_i^\top A_j$  for the new values of  $A_i$  and  $A_j$ . Since we are multiply on the left side for the matrices  $A_{j'}$ 's on line 5 by the same matrix  $\hat{P}$ , this does not break the induction assumption inside the set  $S_j$ . After the update on line 5, for each  $v_{i'} \in S_i$  and each  $v_{j'} \in S_j$ , we have  $A_{i'}^\top A_{j'} = A_{i'}^\top A_i A_i^\top A_j A_j^\top A_{j'} = P_{i'i} P_{ij} P_{jj'} = P_{i'j'}$ . Hence, after line 6 and 7, we know that for each  $v_k \in S'$  (the set defined on the line 6) and each  $v_{i'}, v_{j'} \in S_k$ , we have  $P_{i'j'} = A_{i'}^\top A_{j'}$ . Since the permutation matrices that are changed during this iteration have their corresponding vertices in the set  $S'$ , we know that the induction assumption is correct after this iteration.

From 1, 2 we know that, we could have  $P_{ij} = A_i^\top A_j$  for each  $P_{ij}$  after initialization. Since we have shown in the main paper that the Pairwise Alignment method could solve the problem optimally on this case, we know that our algorithm has also solved the problem optimally after initialization, and hence we do not have any updates in the coordinate update part. Therefore, Algorithm 1 could guarantee to solve the optimization problem optimally under this case.  $\square$

### B. Proof for Theorem 2

*Proof.* Ideally, we want to recover  $(A_1, \dots, A_n)$  so that  $A_i^\top A_j = \hat{T}_{ij}$  for each each pair of  $(A_i, A_j)$ . Let us analyze the probability that we could recover such a tuple of  $(A_1, \dots, A_n)$  under the model in Equation (5).

First, let us consider the probability that we could guarantee to recover the correct permutation matrices  $\hat{T}_{ij}$  from the optimization problem  $\max_P \text{tr}(P^\top T_{ij})$  for any  $i \neq j$ . For any permutation matrix  $P' \in \mathcal{P}_m$ ,  $P' \neq T_{ij}$ , if we denote  $k$  to be the number of entries that  $T_{ij}$  has 1's on but  $P'$  does not, then  $k = \text{tr}((\hat{T}_{ij} - P')^\top \hat{T}_{ij})$ . Therefore,  $U := \frac{\text{tr}(P'^\top T_{ij}) - \text{tr}(\hat{T}_{ij}^\top T_{ij}) + k}{\eta_{ij}}$  follows the Chi-Square

distribution  $\chi^2(2k)$ . Hence, the probability that  $P'$  is a better permutation matrix compared to  $\hat{T}_{ij}$  is

$$\begin{aligned} & \Pr[\text{tr}(P'^\top T_{ij}) \geq \text{tr}(\hat{T}_{ij}^\top T_{ij})] \\ &= \Pr[\eta_{ij}U - k \geq 0] = \Pr\left[\frac{U}{\mathbb{E}[U]} - 1 \geq \frac{1}{2}\left(\frac{1}{\eta_{ij}} - 2\right)\right]. \end{aligned} \quad (7)$$

For  $\eta_{ij} \leq \frac{1}{10}$ , by the Chi-Square tail bounds that Laurent and Massart (2000) proposed,

$$\begin{aligned} & \Pr\left[\frac{U_{ij}}{\mathbb{E}[U_{ij}]} - 1 \geq \frac{1}{2}\left(\frac{1}{\eta_{ij}} - 2\right)\right] \\ & \leq \Pr\left[\frac{U_{ij}}{\mathbb{E}[U_{ij}]} - 1 \geq \frac{1}{4}\left(\frac{1}{\eta_{ij}} - 2\right) + \sqrt{\frac{1}{2}\left(\frac{1}{\eta_{ij}} - 2\right)}\right] \\ & \leq \exp\left(-\frac{k}{4}\left(\frac{1}{\eta_{ij}} - 2\right)\right). \end{aligned} \quad (8)$$

Denote the probability of misaddressing  $k$  letters to  $k$  envelopes (The Bernoulli-Euler Problem of the Misaddressed Letters (Dörrie, 2013)) as  $p_k = \sum_{i=0}^{\infty} \frac{(-1)^i}{i!} \leq \frac{1}{2}$  (for  $k \geq 2$ ). Then by union bound on Equation (8) for  $k = 2, 3, \dots, n$ , we know the probability that there is some  $P' \neq \hat{T}_{ij}$  that is better than  $\hat{T}_{ij}$  is at most

$$\begin{aligned} & \sum_{k=2}^m p_k \cdot \frac{m!}{(m-k)!} \cdot \exp\left(-\frac{k}{4}\left(\frac{1}{\eta_{ij}} - 2\right)\right) \\ & \leq \frac{1}{2} \sum_{k=2}^m m^k \cdot \exp\left(-\frac{k}{4}\left(\frac{1}{\eta_{ij}} - 2\right)\right). \end{aligned} \quad (9)$$

If we have  $\eta_{ij} \leq \frac{1}{4(1+\varepsilon)\ln m+2}$  for some  $\varepsilon > 0$ , therefore,

$$\begin{aligned} & \frac{1}{2} \sum_{k=2}^m m^k \cdot \exp\left(-\frac{k}{4}\left(\frac{1}{\eta_{ij}} - 2\right)\right) \\ & \leq \frac{1}{2} \sum_{k=2}^m m^{-\varepsilon k} = \frac{m^{-2\varepsilon}}{2(1 - m^{-\varepsilon})}. \end{aligned} \quad (10)$$

Hence, if we choose the variance parameter  $\eta_{ij} \leq \min(\frac{1}{10}, \frac{1}{4(1+\varepsilon)\ln m+2})$  for some  $\varepsilon > 0$ , then for  $m \geq 2^{\frac{1}{\varepsilon}}$  we have probability at least  $1 - m^{-2\varepsilon}$  to guarantee that we could recover  $\hat{T}_{ij}$  from the optimization problem  $\max_P \text{tr}(P^\top T_{ij})$ .

Therefore, if we assume that the number of element sets  $n$  is not too large as there exists some constant  $\gamma > 0$  such that  $n \leq m^\gamma$ , then by union bound we know that with probability at least  $1 - m^{-\delta}$  for any  $\delta > 0$  that we could guarantee to use the Pairwise Alignment method to recover a correct solution if  $m \geq 2^{\frac{2}{4\gamma+\delta}}$  and we set each  $\eta_{ij} \leq \min(\frac{1}{10}, \frac{1}{2(2+4\gamma+\delta)\ln m+2}) = O(\frac{1}{\log m})$ .

Then, let us consider the probability that our Algorithm 1 could recover the correct permutation matrices. We could only make some errors on the updates on line 5 and 11. Basically, if we don't make any error at any iteration at the step of computing  $\hat{P}$  on line 4 and don't make any updates on line 11, then we are sure that our algorithm could solve the problem optimally.

Here we consider  $m \geq 8$  such that  $\frac{1}{10} > \frac{1}{4(1+\varepsilon)\ln m+2}$  for any  $\varepsilon > 0$ . Let us first bound the probability that we might make a mistake on computing the matrix  $\hat{P}$  on line 4. At each iteration when we are treating with the edge  $(v_i, v_j)$ , if we have  $\eta_{ij} \leq \frac{1}{4(1+\varepsilon)\ln m+2}$  for any  $\varepsilon > 0$ , then from the above analysis we know that with probability at least  $1 - m^{-2\varepsilon}$  we do not make mistakes on this step.

Otherwise, let us take  $(i^*, j^*) = \operatorname{argmin}_{i' \in S_i, j' \in S_j} \eta_{i'j'}$  ( $S_i$  and  $S_j$  are the sets before being updated on line 7). If we have  $\eta_{i^*j^*} \leq \frac{1}{8(1+\varepsilon)\ln m+4} \leq \frac{1}{4(1+\varepsilon)\ln m+2}$ , then from the above analysis we could also know that with probability at least  $1 - m^{-2\varepsilon}$  we could get  $\hat{T}_{i^*j^*}$  from the optimization problem  $\max_P \operatorname{tr}(P^\top T_{i^*j^*})$ , and also we know that  $\eta_{ij} - \eta_{i^*j^*} \geq \frac{1}{8(1+\varepsilon)\ln m+4}$ .

Notice that  $(v_i, v_j)$  is an edge of the Maximum Spanning Tree of  $G$ , it must be the edge with largest edge weight between vertices in  $S_i$  and  $S_j$ . Therefore, we have  $f(T_{ij}) \geq f(T_{i^*j^*})$ . Conditioned on the cases that we could recover  $\hat{T}_{i^*j^*}$  from  $\max_P \operatorname{tr}(P^\top T_{i^*j^*})$  (we will omit some conditional probability notations from now on for easier illustration, but we will mention with sentences if we are considering some conditional probability), and let  $U \sim \chi^2(m)$  be a Chi-Square random variable with free degree  $m$ , then by the Chi-Square tail bounds that Laurent and Massart (2000) proposed,

$$\begin{aligned} & \Pr[f(T_{i^*j^*}) \leq m(1 - \eta_{i^*j^*} - \frac{1}{16(1+\varepsilon)\ln m+8})] \\ &= \Pr[U - m \geq \frac{m}{\eta_{i^*j^*}(16(1+\varepsilon)\ln m+8)}] \\ &\leq \Pr[U - m \geq \frac{m}{2}] \leq \Pr[U - m \geq 0.48m] \\ &\leq \exp(-\frac{m}{25}) \leq m^{-2\varepsilon} \end{aligned} \tag{11}$$

for sufficiently large  $m$ . On the other side, consider the value of  $f(T_{ij})$ , denote  $P' = \operatorname{argmax}_P \operatorname{tr}(P^\top T_{ij})$  and  $k$  to be the number of entries that  $\hat{T}_{ij}$  has 1's on, but  $P'$  does not ( $0 \leq k \leq m$ ). Since we require all  $\eta_{ij} \leq O(1)$ , let us assume that we have  $\eta_{ij} \leq \frac{1}{3}$ . Let  $V_1 \sim \chi^2(k)$ ,  $V_2 \sim \chi^2(m-k)$  be two independent Chi-Square random variables (we use  $\chi^2(0)$  to be the random variable only has support on a single point 0). Conditioned on  $k$ , the distribution of  $f(T_{ij})$  is the same with  $\eta_{ij}(V_1 - V_2) + m - k$ . If  $k > 0$ , we know that

$$\begin{aligned} & \Pr[V_1 \geq k + \frac{m}{\eta_{ij}(32(1+\varepsilon)\ln m+16)}] \\ &\leq \Pr[V_1 - k \geq \frac{3m}{32(1+\varepsilon)\ln m+16}] \\ &\leq \Pr[V_1 - k \geq 2\sqrt{2k\varepsilon\ln m} + 4\varepsilon\ln m] \leq m^{-2\varepsilon} \end{aligned} \tag{12}$$

for sufficiently large  $m$ . Symmetrically, if  $k < m$ ,

$$\begin{aligned} & \Pr[V_2 \leq (m-k) - \frac{n}{\eta_{ij}(32(1+\varepsilon)\ln m+16)}] \\ &\leq \Pr[(m-k) - V_2 \geq \frac{3m}{32(1+\varepsilon)\ln m+16}] \\ &\leq \Pr[(m-k) - V_2 \geq 2\sqrt{2k\varepsilon\ln m}] \leq m^{-2\varepsilon}. \end{aligned} \tag{13}$$

for sufficiently large  $m$ . Therefore, conditioned on  $k$ , if we have  $\eta_{ij} \leq \frac{1}{3}$ , we always have



$$\begin{aligned}
& \Pr[\eta_{ij}(V_1 - V_2) + m - k \geq m(1 - \eta_{ij} + \frac{1}{16(1+\varepsilon)\ln m + 8})] \\
& \leq \Pr[\eta_{ij}(V_1 - V_2) - (2k - m)\eta_{ij} \geq \frac{1}{16(1+\varepsilon)\ln m + 8}] \\
& \leq \Pr[V_1 \geq k + \frac{m}{\eta_{ij}(32(1+\varepsilon)\ln m + 16)}] \\
& + \Pr[V_2 \leq (m - k) - \frac{m}{\eta_{ij}(32(1+\varepsilon)\ln m + 16)}] \leq 2m^{-2\varepsilon}.
\end{aligned} \tag{14}$$

This is true for all  $k$ . Hence, without conditioned on  $k$ , we know that

$$\Pr[f(T_{ij}) \geq m(1 - \eta_{ij} + \frac{1}{16(1+\varepsilon)\ln m + 8})] \leq 2m^{-2\varepsilon} \tag{15}$$

for sufficiently large  $m$  and if we have  $\eta_{ij} \leq \frac{1}{3}$ .

By union bound on Equations (11) and (15), we know that, conditioned on the cases that we could recover  $\hat{T}_{i^*j^*}$  from  $\max_P \text{tr}(P^\top T_{i^*j^*})$ , since  $\eta_{ij} - \eta_{i^*j^*} \geq \frac{1}{8(1+\varepsilon)\ln m + 4}$ , we have

$$\begin{aligned}
& \Pr[f(T_{ij}) \geq f(T_{i^*j^*})] \\
& \leq \Pr[f(T_{i^*j^*}) \leq m(1 - \eta_{i^*j^*} - \frac{1}{16(1+\varepsilon)\ln m + 8})] \\
& + \Pr[f(T_{ij}) \geq m(1 - \eta_{ij} + \frac{1}{16(1+\varepsilon)\ln m + 8})] \\
& \leq 3m^{-2\varepsilon}.
\end{aligned} \tag{16}$$

Since we know that, if we have  $\eta_{i^*j^*} \leq \frac{1}{8(1+\varepsilon)\ln m + 4}$ , then with probability at least  $1 - m^{-2\varepsilon}$  we could recover  $\hat{T}_{i^*j^*}$  from  $\max_P \text{tr}(P^\top T_{i^*j^*})$ . Hence, conditioned on the case that  $\eta_{ij} > \frac{1}{4(1+\varepsilon)\ln m + 2}$ , we know that the probability  $\Pr[f(T_{ij}) \geq f(T_{i^*j^*})] \leq 3m^{-2\varepsilon} + m^{-2\varepsilon} = 4m^{-2\varepsilon}$ . Plus the opposite case where  $\eta_{ij} \leq \frac{1}{4(1+\varepsilon)\ln m + 2}$ , by union bound we know that the probability that we could make error during each iteration of the initialization part of Algorithm 1 is at most  $5m^{-2\varepsilon}$ . This is true under the condition that  $\eta_{ij} \leq \frac{1}{3}$  and  $\min_{i' \in S_i, j' \in S_j} \eta_{ij} \leq \frac{1}{8(1+\varepsilon)\ln m + 4}$ . To make these two conditions true, we could make the following two requirements:

- Consider an undirected weighted graph  $G' = (V', E'')$ , where there is a vertex  $v'_i$  for each element set  $X_i$  and there is an edge  $(v'_i, v'_j) \in E''$  with edge weight  $\eta_{ij}$ . Then the bottleneck weight of the Minimum bottleneck spanning tree of  $G'$  should be at most  $\frac{1}{8(1+\varepsilon)\ln m + 4}$ .
- $\max_{i < j} \eta_{ij} \leq \frac{1}{3}$ .

Therefore, under the above two conditions, assume the number of element sets  $m$  satisfy  $n \leq m^\gamma$  for some constant  $\gamma > 0$ , then by union bound we know that, for sufficiently large  $n$ , The probability that we could recover the a correct solution for  $(\hat{A}_1, \dots, \hat{A}_n)$  during the initialization part of the Algorithm 1 with probability at  $1 - 5m^{-2\varepsilon+\gamma}$ .

For the coordinate update part of Algorithm 1 (line 9 to 12), let us consider the probability that we do not do any updates there conditioned on the case that we have already get an optimal solution in the initialization part. For each step, denote the matrix we are optimizing as  $A_i$ . The update rule is Equation (3). Using the same idea we have used before, assume there is some matrix  $P' \neq A_i$

such that  $\text{tr}(P'^\top \sum_{1 \leq j \leq n, i \neq j} A_j T_{ij}) \geq \text{tr}(A_i^\top \sum_{1 \leq j \leq n, i \neq j} A_j T_{ij})$ . Denote  $k$  as the number of entries that  $A_i$  has 1's on, but  $P'$  does not. Also, denote  $U_1, \dots, U_{i-1}, U_{i+1}, \dots, U_n$  to be independent random variables following the distribution  $\chi^2(2k)$ , and let  $U$  be a random variable following distribution  $\chi^2(2k(n-1))$ . Then, by the Chi-Square tail bounds that Laurent and Massart (2000) proposed

$$\begin{aligned} & \Pr[\text{tr}(P'^\top \sum_{1 \leq j \leq n, i \neq j} A_j T_{ij}) \geq \text{tr}(A_i^\top \sum_{1 \leq j \leq n, i \neq j} A_j T_{ij})] \\ &= \Pr[\sum_{1 \leq j \leq n, i \neq j} \eta_{ij} U_j \geq k(n-1)] \\ &\leq \Pr[\frac{1}{3}U \geq k(n-1)] \leq \exp(-\frac{2k(n-1)}{25}) \end{aligned} \tag{17}$$

Then, again by union bound on all values for  $k$ , we know that the probability that we might get a wrong answer for  $A_i$  in a single step is at most

$$\begin{aligned} & \sum_{k=2}^m p_k \cdot \frac{m!}{(m-k)!} \cdot \exp(-\frac{2k(n-1)}{25}) \\ &\leq \frac{1}{2} \sum_{k=2}^m m^k \cdot \exp(-\frac{2k(n-1)}{25}) \end{aligned} \tag{18}$$

If we have  $n \geq 20 \ln m$  for some  $\varepsilon' > 0$ , then for sufficient large value of  $m$  we have

$$\frac{1}{2} \sum_{k=2}^m m^k \cdot \exp(-\frac{2k(n-1)}{25}) \leq m^{-2}. \tag{19}$$

By union bound on all  $m$  matrices  $A_i$ 's, we know that the probability of at least one of them needs updates is at most  $m^{-1}$ . Hence, the probability that we could solve the optimization problem with probability at least  $1 - 5m^{-2\varepsilon+\gamma} + m^{-1}$  under all of the above constraints. If we set  $\varepsilon = \frac{\gamma+1}{2}$ , then the probability becomes  $1 - 6m^{-1} = 1 - o(1)$  for sufficiently large  $m$ . Under that setting, we require the bottleneck weight of the Minimum bottleneck spanning tree of  $G'$  to be at at most  $\frac{1}{8(1+\varepsilon) \ln m + 4} = \frac{1}{4(3+\gamma) \ln m + 4}$ .  $\square$